# OOP Basics Exam – Animal Centre



### Overview

In this exam, you need to build an **animal centre** project, which has support for **animals**, **procedures** for storing procedures of animals,and a **hotel** for storing animals in the animal center. The project will consist of **model classes** and a **controller class**, which manages the **interaction** between the **animals**, **procedures** and **hotel**.

### Setup

To set up your project, create a new Visual Studio project with the name **"**AnimalCentre**".** The project must have a StartUp class with the namespace **"**AnimalCentre**".** You are free to use any namespaces you want, **as long as you have a class, called** StartUp **in the** AnimalCentre **namespace**. **Not following this rule will lead to your code not compiling in the Judge system**.

## Task 1: Structure (50 points)

You are given **3** interfaces which you have to implement **by yourself** and then implement their functionality in the **correct classes**.

There are **3** types of entities in the application: **Animals**, **Procedures** and **Hotel**:

### Animals

The Animal is a **base class** for any **animal** and it **should not be able to be instantiated**.

#### Data

* Name – **string**
* Happiness – **int (can’t be less than 0 or more than 100. In these cases throw ArgumentException with message "**Invalid happiness**")**
* **Energy – int (can’t be less than 0 or more than 100. In these cases throw ArgumentException with message "**Invalid energy**")**
* **ProcedureTime – int**
* **Owner – string (by default: "**Centre**")**
* **IsAdopt – bool (by default: false)**
* **IsChipped – bool (by default: false)**
* **IsVaccinated – bool (by default: false)**

#### Constructor

An **animal** should take the following values upon initialization:

string name, int energy, int happiness, int procedureTime

#### Child Classes

There are several concrete types of **animals**:

* Cat
* Dog
* Lion
* Pig

**Each of these concrete types have a** ToString() **method in the format:**

**"** Animal type: {animal type} - {animal name} - Happiness: {animal happiness} - Energy: {animal energy}**"**

### Procedures

The Procedure is a **base class** for any **procedures** and it **should not be able to be instantiated**.

#### Data

* ProcedureHistory – collection of **Animals** accessible only by the child classes(will hold information about each procedure and its animals)

#### Constructor

A **procedure** should not take any values upon initialization

#### Behavior

##### string History()

Returns a string with information about **each procedure** and its animals. The returned string must be in the following format:

**"**{procedure name}**"**

**"** - {animal name} - Happiness: {happiness} - Energy: {energy}**"**

**void DoService(IAnimal animal, int procedureTime)**

**Each procedure** implements its own DoService()method with different logic, which is explained below.

#### Child Classes

There are several concrete types of **procedures**, which execute **different logic** when DoService() is called:

* Chip – removes **5** happiness and chips current animal. Animal can be chipped once. If animal is already chipped throw an ArgumentException with message **"**{animal name} is already chipped**"**
* DentalCare – adds **12** happiness and **10** energy
* Fitness – removes **3** happiness and adds **10** energy
* NailTrim – removes **7** happiness
* Play – removes **6** energy and adds **12** happiness
* Vaccinate – removes **8** energy and **vaccinates** current animal

Each procedure must check if the animal procedure time is **more than or equal to** the time each procedure will take. If animal procedure time is lower than the time for the current procedure throw ArgumentException with message **"**Animal doesn't have enough procedure time**"**

Every time a procedure command is called, the time the procedure took is **subtracted** from the **animal’s** **allowed procedure time**.

### Hotel

The Hotel is a **class** which **should be able to be instantiated**.

The **hotel** is a building, which holds **animals**.

#### Data

* Capacity – **int** with a constant value of **10**
* Animals – Collection with the animal’s **name** as the **key** and the **animal itself** as the **value**

#### Constructor

A **hotel** should not take any values upon initialization:

#### Behavior

##### void Accommodate(IAnimal animal)

If there isn’t enough capacity in the hotel throw an InvalidOperationException with the message **"**Not enough capacity**"**

If an animal with this name already exist in the hotel, throw an ArgumentException with the message **"**Animal {animal name} already exist**"**

In any other case, add the current animal to the hotel.

##### void Adopt(string animalName, string owner)

If the provided animal name does not exist in the hotel, throw an ArgumentException with the message **"**Animal {animal name} does not exist**"**

If an animal with that name exist, **change its owner**, its **adoption status** and **remove the animal from the hotel**.

## Task 2: Business Logic (150 points)

### The Controller Class

The business logic of the program should be concentrated around several **commands**. Implement a class called AnimalCentre, which will hold the **main functionality**.

***Note: The*** *AnimalCentre* ***class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!***

The main functionality is represented by these **public** **methods**:

|  |
| --- |
| AnimalCentre.cs |
| public string RegisterAnimal(string type, string name, int energy, int happiness, int procedureTime)  {  throw new **NotImplementedException**();  }  public string Chip(string name, int procedureTime)  {  throw new **NotImplementedException**();  }  public string Vaccinate(string name, int procedureTime)  {  throw new **NotImplementedException**();  }  public string Fitness(string name, int procedureTime)  {  throw new **NotImplementedException**();  }  public string Play(string name, int procedureTime)  {  throw new **NotImplementedException**();  }  public string DentalCare(string name, int procedureTime)  {  throw new **NotImplementedException**();  }  public string NailTrim(string name, int procedureTime)  {  throw new **NotImplementedException**();  }  public string Adopt(string animalName, string owner)  {  throw new **NotImplementedException**();  }  public string History(string type)  {  throw new **NotImplementedException**();  } |

**NOTE: The** AnimalCentre **class should not handle any exceptions. That should be the responsibility of the class, which reads the commands and passes them to the** AnimalCentre**.**

### Commands

There are several commands which control the business logic of the application. They are stated below.

**NOTE:** For each command except for **"**RegisterAnimal**"**, you must check if an animal with that name exist in the hotel. If it doesn’t, throw an ArgumentException with the message **"**Animal {animal name} does not exist**"**.

#### RegisterAnimal Command

##### Parameters

* AnimalType – string
* Name – string
* Energy – int
* Happiness – int
* ProcedureTime – int

##### Functionality

Creates an animal with the correct type and **registers** it in the hotel

If the animal happiness or energy are invalid throw ArgumentException for Invalid energy/happiness.

If the hotel capacity is not enough, don’t register the animal and throw InvalidOperationException, with the correct message for capacity.

If an animal with the **same name** already exist in the hotel, don’t register it and throw ArgumentException with message **"**Animal {animal name} already exist**"**

If successful, returns "Animal {animal name} registered successfully".

#### Chip Command

##### Parameters

* Name - string
* ProcedureTime - int

##### Functionality

Calls the Chip procedure with parameters **currentAnimal** and **procedureTime**.

Returns "{current animal name} had chip procedure".

#### Vaccinate Command

##### Parameters

* Name - string
* ProcedureTime - int

##### Functionality

Calls the Vaccinate procedure with parameters currentAnimal and procedureTime.

Returns "{current animal name} had vaccination procedure".

#### Fitness Command

##### Parameters

* Name - string
* ProcedureTime - int

##### Functionality

Calls the Fitness procedure with parameters currentAnimal and procedureTime.

Returns "{current animal name} had fitness procedure".

#### Play Command

##### Parameters

* Name - string
* ProcedureTime - int

##### Functionality

Calls the Play procedure with parameters currentAnimal and procedureTime.

Returns "{current animal name} was playing for {procedure time} hours".

#### DentalCare Command

##### Parameters

* Name - string
* ProcedureTime - int

##### Functionality

Calls the DentalCare procedure with parameters currentAnimal and procedureTime.

Returns "{current animal name} had dental care procedure".

#### NailTrim Command

##### Parameters

* Name - string
* ProcedureTime - int

##### Functionality

Calls the NailTrim procedure with parameters currentAnimal and procedureTime.

Returns "{current animal name} had nail trim procedure".

#### Adopt Command

##### Parameters

* animalName - string
* ownerName - string

##### Functionality

Finds the animal with that name in the hotel and adopts it.

Returns:

If the current animal **is** chipped:

**"**{owner} adopted animal with chip**"**

If the current animal **is not** chipped:

**"**{owner} adopted animal without chip**"**

#### History Command

##### Parameters

* procedureType - string

##### Functionality

Returns information about **all animals** which had current procedure type in the format:

"{procedure type}"

" Animal type: {animal type} - {animal name} - Happiness: {animal happiness} - Energy: {animal energy}"

## Task 3: Input / Output

### Input

* You will receive commands **until you receive “End”** as a command.

Below, you can see the **format** in which **each command** will be given in the input:

* RegisterAnimal {type} {name} {energy} {happiness} {procedureTime}
* Chip {name} {procedureTime}
* Vaccinate {name} {procedureTime}
* Fitness {name} {procedureTime}
* Play {name} {procedureTime}
* DentalCare {name} {procedureTime}
* NailTrim {name} {procedureTime}
* Adopt {animal name} {owner}
* History {procedureType}

### Output

Print the output from each command when issued. When the end command is received, print the information about all adopted animals and their owners **by order of name alphabetical** in the format:

"--Owner: {owner name}

- Adopted animals: {animals name separated by space}"

If an InvalidOperationException is thrown during any of the commands’ execution, print:

"InvalidOperationException: " plus the message of the exception

If an ArgumentException is thrown during any of the commands’ execution, print:

"ArgumentException: " plus the message of the exception

### Constraints

* The commands will always be in the provided format.

### Examples

|  |
| --- |
| **Input** |
| RegisterAnimal Lion Gogo 30 50 6  Chip Gogo 3  Adopt Gogo Pesho  RegisterAnimal Pig Mimi 20 40 5  Chip Mimi 3  Play Mimi 1  RegisterAnimal Lion Gogo 30 50 6  NailTrim Mimi 4  DentalCare Mimi 6  History Chip  Adopt Mimi Pesho  End |
| **Output** |
| Animal Gogo registered successfully  Gogo had chip procedure  Pesho adopted animal with chip  Animal Mimi registered successfully  Mimi had chip procedure  Mimi was playing for 1 hours  Animal Gogo registered successfully  ArgumentException: Animal doesn't have enough procedure time  ArgumentException: Animal doesn't have enough procedure time  Chip  Animal type: Lion - Gogo - Happiness: 45 - Energy: 30  Animal type: Pig - Mimi - Happiness: 47 - Energy: 14  Pesho adopted animal with chip  --Owner: Pesho  - Adopted animals: Gogo Mimi |

|  |
| --- |
| **Input** |
| RegisterAnimal Dog Gogo 30 50 6  Chip Gogo 3  Adopt Gogo Pesho  RegisterAnimal Pig Mimi 20 40 5  Chip Mimi 3  Play Mimi 1  RegisterAnimal Lion Gogo 30 50 6  RegisterAnimal Cat Maca 101 50 6  RegisterAnimal Cat Maca 20 50 8  Vaccinate Maca 4  Adopt Maca Ani  History Vaccinate  End |
| **Output** |
| Animal Gogo registered successfully  Gogo had chip procedure  Pesho adopted animal with chip  Animal Mimi registered successfully  Mimi had chip procedure  Mimi was playing for 1 hours  Animal Gogo registered successfully  ArgumentException: Invalid energy  Animal Maca registered successfully  Maca had vaccination procedure  Ani adopted animal without chip  Vaccinate  Animal type: Cat - Maca - Happiness: 50 - Energy: 12  --Owner: Ani  - Adopted animals: Maca  --Owner: Pesho  - Adopted animals: Gogo |

|  |
| --- |
| **Input** |
| RegisterAnimal Lion Maya 10 20 14  RegisterAnimal Lion Chico 20 40 15  RegisterAnimal Lion Mo 10 20 14  RegisterAnimal Lion Tosho 20 40 15  RegisterAnimal Lion Maya 10 20 14  RegisterAnimal Lion Koko 20 40 15  RegisterAnimal Lion Zuzi 10 20 14  RegisterAnimal Lion Gazi 20 40 15  RegisterAnimal Lion Chibo 10 20 14  RegisterAnimal Lion Fifi 20 40 15  RegisterAnimal Lion Kiki 10 20 14  RegisterAnimal Lion Toni 20 40 15  End |
| **Output** |
| Animal Maya registered successfully  Animal Chico registered successfully  Animal Mo registered successfully  Animal Tosho registered successfully  ArgumentException: Animal Maya already exist  Animal Koko registered successfully  Animal Zuzi registered successfully  Animal Gazi registered successfully  Animal Chibo registered successfully  Animal Fifi registered successfully  Animal Kiki registered successfully  InvalidOperationException: Not enough capacity |

## Task 4: Unit testing (100 points)

You will receive a skeleton with one class inside. The class will have some methods, properties, fields and constructors. Cover the whole class with unit test to make sure that the class is working as intended.